EASEL Version 1.1
for OS/2
Extended Edition

Developer's Guide Supplement
Dynamic Data Exchange Support

EASEL Version 1.1
for OS/2
Extended Edition

Developer's Guide Supplement
Dynamic Data Exchange Support

First Edition (August 1990)

This publication could include technical inaccuracies or typographical errors. Changes are made periodically to the information herein; these changes will be incorporated in new editions of this publication.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's licensed program may be used. Any functionally equivalent program may be used instead.

Requests for copies of this product and for technical information about the system should be made to your local IBM representative.

© Copyright Interactive Images, Inc. 1989, 1990. All rights reserved.

Note to US Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

IBM and OS/2 are registered trademarks of International Business Machines Corporation in the United States and other countries. EASEL is a registered trademark of Interactive Images, Inc. in the United States and other countries.

Preface

This publication is a supplement to the *IBM EASEL® Version 1.1 for OS/2® Extended Edition Developer's Guide*. It describes the IBM EASEL for OS/2 Extended Edition (referred to as EASEL OS/2 EE) support for the Operating System/2™ (OS/2) Presentation Manager™ (PM) Dynamic Data Exchange (DDE) protocol.

Additional DDE protocol information is provided in the *IBM OS/2 Programming Tools and Information* publications.

EASEL OS/2 EE is developed by Interactive Images, Inc., Woburn, MA.

IBM and OS/2 are registered trademarks of International Business Machines Corporation in the United States and other countries.

EASEL is a registered trademark of Interactive Images, Inc. in the United States and other countries.

Operating System/2 and Presentation Manager are trademarks of International Business Machines Corporation in the United States and other countries.

Contents

1. DDE Support 1-1
1.1 The DDE Protocol 1-2
1.2 The DDE Stimulus Library 1-3
1.2.1 Conversation Identifier 1-3
1.2.2 PM Message 1-4
1.2.3 Include File 1-4
1.3 Overview of a DDE Conversation 1-5
1.3.1 Client and Server as DDE Partners 1-5
1.3.1.1 EASEL OS/2 EE Program as a DDE Partner 1-5
1.3.1.2 Non-EASEL OS/2 EE Program as Server or Client 1-5
1.3.1.3 EASEL OS/2 EE Program as a Client 1-5
1.3.1.4 EASEL OS/2 EE Program as a Server 1-6
1.3.1.5 Requests for Data 1-6
1.3.1.6 Advise State 1-6
1.3.1.7 Terminating a Conversation 1-6
1.3.2 Responding to DDE Events 1-7
1.3.2.1 DDE Protocol Rules 1-7
1.3.2.2 Acknowledgments 1-7
1.3.2.3 Events Requiring Client Response 1-7
1.3.2.4 Events Requiring Server Response 1-8
1.3.3 Sample Programs 1-8
1.3.3.1 Client Program 1-9
1.3.3.2 Server Program 1-11
1.4 The DDE Functions and Subroutines 1-14
1.4.1 Summary of Functions and Subroutines 1-14
1.4.2 Polling For Topics 1-18
1.4.2.1 DDEPollTopics Function 1-19
1.4.2.2 DDEEnumeratePollTopics Function 1-20
1.4.2.3 DDEGetPollAppName Function 1-21
1.4.2.4 DDEGetPollTopic Function 1-22
1.4.3 Starting Conversations 1-23
1.4.3.1 DDEInitiate Function 1-24
1.4.3.2 DDERegisterServer Subroutine 1-25
1.4.3.3 DDEUnRegisterServer Subroutine 1-27
1.4.3.4 DDEEnumerateConversations Function 1-28
1.4.4 Exchanging Data 1-29
1.4.4.1 DDERequest Subroutine 1-30
1.4.4.2 DDEDataString Subroutine 1-31
1.4.4.3 DDEExecute Subroutine 1-33
1.4.4.4 DDEAdvise Subroutine 1-34
1.4.4.5 DDEUnAdvise Subroutine 1-36
1.4.4.6 DDEPokeString Subroutine 1-37
1.4.4.7 DDEAck Subroutine 1-39
1.4.5 Terminating Conversations 1-40
1.4.5.1 DDETerminate Subroutine 1-41
1.4.5.2 DDETerminateAll Subroutine 1-42

1.4.6 Getting Information on Existing Conversations 1-43
1.4.6.1 DDEGetAppName Function 1-45
1.4.6.2 DDEGetTopic Function 1-46
1.4.6.3 DDEGetFormat Function 1-47
1.4.6.4 DDEGetItemName Function 1-48
1.4.6.5 DDEGetStringData Function 1-49
1.4.6.6 DDEGetStatus Function 1-50
1.4.6.7 DDEGetErrorID Function 1-51
1.4.7 Manipulating Status Variables 1-52
1.4.7.1 DDESetStatusFlag Function 1-53
1.4.7.2 DDEQueryStatusFlag Function 1-54
1.4.8 Miscellaneous Calls 1-55
1.4.8.1 DDERegisterFormat Subroutine 1-56
1.4.8.2 DDEGetPID Function 1-57
1.4.8.3 DDEGetErrorString Function 1-58
1.4.8.4 DDESetTraceLevel Subroutine 1-59
1.5 The DDE Events 1-60
1.5.1 DDE Default Processing 1-60
1.5.2 Summary of DDE Events 1-61
1.5.3 DDE Events List 1-63
1.5.3.1 DDE_INITIATE 1-64
1.5.3.2 DDE_INITIATEACK 1-65
1.5.3.3 DDE_REQUEST 1-66
1.5.3.4 DDE_ACK 1-68
1.5.3.5 DDE_NACK 1-69
1.5.3.6 DDE_DATA 1-70
1.5.3.7 DDE_ADVISE 1-72
1.5.3.8 DDE_UNADVISE 1-75
1.5.3.9 DDE_POKE 1-77
1.5.3.10 DDE_EXECUTE 1-79
1.5.3.11 DDE_TERMINATE 1-80
1.5.3.12 DDE_ERROR 1-81
1.5.4 Recognizing DDE Errors 1-82
1.5.5 DDE Error Messages 1-82
1.6 DDE Support Notes 1-84
1.6.1 System Topic 1-84
1.6.2 Guarded Blocks 1-84
1.6.3 EASEL OS/2 EE DDE Conversation 1-84
1.6.4 Maximum Length of Data 1-84
1.7 DDE Tables 1-85
1.7.1 Functions and Subroutines 1-86
1.7.2 DDE Events 1-87
1.7.3 Client Functions/Subroutines and Events 1-88
1.7.4 Server Functions/Subroutines and Events 1-89
1.7.5 PM Messages 1-90
1.8 DDE Include File 1-91

Index X-1

1. DDE Support

1.1 The DDE Protocol

Dynamic Data Exchange (DDE) is a message-based protocol that allows two OS/2 Presentation Manager™ (PM) applications to establish a connection through which they can exchange data, or commands, in any format that they mutually agree to support. Only applications that are specifically implemented to support DDE can establish such a connection.

The basic conceptual model for a DDE connection is a conversation. A DDE participant can be a client or a server:

Client — Initiates a conversation; controls the flow of the conversation.

Server — Responds to an initiation; responds to requests from the client.

Each conversation has an associated topic name, which is established at the time the conversation is started. Once a conversation is established, it is the client that controls the flow.

The client does any or all of the following:

- Issues one or more requests for data from a server.
- Sends unsolicited data to a server.
- Asks a server to execute commands.
- Establishes an advise state (also known as a hot link) with a server. An advise state is one where the client receives notification (and possibly data) from the server each time some data is modified.

Each individual data exchange within a conversation has an associated item name and format. These are established at the time the data is requested, and can be any name and format on which the two applications agree.

The protocol contains provisions that allow the server to reject requests. Several requests for the same data may be required before the two partners agree on a data format that both can handle. Either partner can terminate the conversation at any time.

Although a single conversation involves only two participants, an individual application can carry on multiple conversations with one or more partner applications, presenting itself as a client, a server, or both.

Presentation Manager is a trademark of International Business Machines Corporation in the United States and other countries.

1.2 The DDE Stimulus Library

EASEL OS/2 EE provides support for DDE via an EASEL OS/2 EE stimulus library. A stimulus library is a specially written dynamic link library (DLL). Dynamic link libraries are discussed in Section 25.2, Writing Library Functions and Subroutines, of the *IBM EASEL Version 1.1 for OS/2 Extended Edition Developer's Guide*. Stimulus libraries are discussed in Section 25.3, Writing a Stimulus Library, of the *IBM EASEL Version 1.1 for OS/2 Extended Edition Developer's Guide*.

The EASEL OS/2 EE DDE stimulus library:

- Contains 30 functions and subroutines that an EASEL OS/2 EE program can call. These allow an EASEL OS/2 EE program to post OS/2 PM DDE messages and manage DDE conversations.
- Defines 12 stimulus events that an EASEL OS/2 EE program can invoke. These allow an EASEL OS/2 EE program to be notified of receipt of OS/2 PM DDE messages.

The EASEL OS/2 EE program can participate in one or more DDE conversations, either as a client, a server, or both. In many cases, the library provides a default behavior for DDE messages that allows the EASEL OS/2 EE program to ignore many of the low level details of the DDE protocol. Many EASEL OS/2 EE programs can participate in a DDE conversation by calling only a few of the DDE functions, and by responding to only a few of the DDE events. In any case, either one or both of the participating applications can be EASEL OS/2 EE programs.

1.2.1 Conversation Identifier

Each conversation in which an EASEL OS/2 EE program engages is identified by a unique conversation identifier. A conversation identifier is an EASEL OS/2 EE integer that is:

- Used as an argument in many of the EASEL OS/2 EE DDE DLL subroutine and function calls.
- Passed as the value of **eventparam** in all of the DDE stimulus library events, if the conversation has been successfully initiated.

The identifier is assigned by the EASEL OS/2 EE DDE DLL.

1.2.2 PM Message

Since DDE is a PM message-based protocol, this document sometimes refers to PM message names (for example, WM_DDE_INITIATE). This is so that you can know how EASEL OS/2 EE DDE support translates into PM messages. Most of the EASEL OS/2 EE DDE events correspond directly to receipt of the like-named PM DDE message, and many of the EASEL OS/2 EE DDE subroutines post or send a PM DDE message. You can always distinguish the PM message names from EASEL OS/2 EE DDE events by noting that the PM message names begin with the letters "WM_".

1.2.3 Include File

The EASEL OS/2 EE DDE DLL comes with an include file (ESLDDE.INC) that contains:

- Declarations of all DDE functions and subroutines.
- Constant definitions for all **eventnumber** values.
- Constant definition for the default format.

Additionally, since many of the DDE functions require a status parameter that is used as part of the DDE protocol, ESLDDE.INC contains:

- Constant definitions for status values.
- Declarations for two functions used to compose and decompose status values.

1.3 Overview of a DDE Conversation

1.3.1 Client and Server as DDE Partners

1.3.1.1 EASEL OS/2 EE Program as a DDE Partner

An EASEL OS/2 EE program that wants to be a DDE partner must minimally satisfy three requirements:

- The program must declare the DDE stimulus library. The library is called "**esldde**".

  Note: An EASEL OS/2 EE program should not contain more than one declaration for the DDE stimulus library using different logical names.

- The program must establish itself as a client, a server, or both.
- The program must contain responses to the appropriate DDE events.

1.3.1.2 Non-EASEL OS/2 EE Program as Server or Client

If the client or server is a non-EASEL OS/2 EE program, read that program's documentation to determine how and when it establishes itself as a client or server.

1.3.1.3 EASEL OS/2 EE Program as a Client

An EASEL OS/2 EE program establishes itself as a client by:

1. Calling **DDEInitiate** one or more times. (The details of the calls and stimulus events discussed in this section are described in 1.4, "The DDE Functions and Subroutines" on page 1-14, and in 1.5, "The DDE Events" on page 1-60.)
2. Passing the DDE application/topic name pair registered by the server and with which it wants to converse.

The application name and topic name must be provided by the server manufacturer. Note that the application name does not necessarily correspond to the executable name. If the client does not know the names, then the EASEL OS/2 EE DDE DLL provides services for polling for all available servers. (See 1.4.2.1, "DDEPollTopics Function" on page 1-19.)

Once a partner responds, the conversation is established immediately, and a conversation ID is returned to the client.

1.3.1.4 EASEL OS/2 EE Program as a Server

An EASEL OS/2 EE program establishes itself as a server by:

1. Calling **DDERegisterServer**.
2. Passing the DDE application/topic name pair by which it is known.

No conversation is established directly as a result of this call, since it is a client that initiates a conversation. A server conversation is not established until a client requests a conversation with the application/topic pair registered by that server. When this happens, a DDE stimulus event is invoked in the server program.

1.3.1.5 Requests for Data

Once a conversation is established, a client can make one or more requests for data in a particular format through the EASEL OS/2 EE DDE DLL subroutine calls, and can respond to the events invoked upon receipt of that data.

A server will generally not voluntarily call DDE DLL subroutines and functions to send data; instead it will only do so in response to events invoked when a client partner requests data.

1.3.1.6 Advise State

An exception to this rule is a server that is capable of supporting an advise state or hot link (notifying a client whenever some data changes). If such a state is established, a server will also call DDE DLL functions and subroutines whenever it recognizes that data that is the subject of an ongoing advise state has changed.

1.3.1.7 Terminating a Conversation

At any point, either of the two partners can terminate the conversation. Once a conversation is terminated, no other transactions using the conversation identifier for that conversation are legal.

1.3.2 Responding to DDE Events

1.3.2.1 DDE Protocol Rules

The DDE protocol defines certain rules about how two applications:

- Engage in a conversation.
- Exchange data throughout a conversation.
- Terminate a conversation.

1.3.2.2 Acknowledgments

Most events require the receiving partner to acknowledge receipt of that event. Acknowledgments are either:

- Positive, meaning the partner accepted the data or can honor the request, or
- Negative, meaning the partner could not honor or understand the request or the data it received.

The acknowledgment requirements are listed on the following page, with the description of each event. (See 1.5, "The DDE Events" on page 1-60 for detailed discussions of these DDE events). Likewise, each subroutine/function description in 1.4, "The DDE Functions and Subroutines" on page 1-14 describes the response that the caller of that routine can expect based on the protocol.

Upon receipt of an event, the correct acknowledgment or required response must be carried out *before* another DDE exchange is initiated. When calling a subroutine or function that transmits an event to the partner, the transmitting program should *not* call another subroutine or function that transmits an event to the partner until the partner has acknowledged the first event. It is very important that any application that participates in DDE follow these rules. Failure to do so will yield unpredictable results.

1.3.2.3 Events Requiring Client Response

In general, a client will need to respond to the following events:

Event	Description
DDE_INITIATEACK	Server is acknowledging that a conversation is established.
DDE_DATA	Data has arrived from the partner.
DDE_ACK	Server is acknowledging a request (positive).
DDE_NACK	Server is acknowledging a request (negative).

DDE_TERMINATE	Server is terminating.
DDE_ERROR	An error has been encountered.

1.3.2.4 Events Requiring Server Response

A server will need to respond to the following events:

Event	Description
DDE_INITIATE	A new conversation has been established.
DDE_REQUEST	Partner is requesting data.
DDE_ACK	Partner is acknowledging receipt of data (positive).
DDE_NACK	Partner is acknowledging receipt of data (negative).
DDE_TERMINATE	Partner is terminating.
DDE_ERROR	An error has been encountered.

Some servers will also respond to the remaining events, in addition to, or in place of DDE_REQUEST:

Event	Description
DDE_EXECUTE	Client wants commands executed.
DDE_POKE	Client is sending data *to* the server.
DDE_ADVISE	Client wants to establish an advise state.
DDE_UNADVISE	Client wants to terminate a hot link.

1.3.3 Sample Programs

The following two sample programs represent

- A sample DDE client that initiates a single DDE conversation consisting of a single request for data.
- A sample DDE server that accepts a single DDE conversation on a single application/topic pair.

These programs are distributed with EASEL OS/2 EE, as follows:

Client Program: DDECSAMP.EAL

Server Program: DDESSAMP.EAL

1.3.3.1 Client Program

```
#####################################################################
# This program is a sample DDE client that initiates a single DDE
# conversation with Microsoft Excel.  The conversation consists of a single
# request for a copy of row 1 from the spreadsheet "sheet1.xls".
# It assumes that Excel is already running, and that "sheet1.xls" is open.
#
# If the initial value of the AppName variable in this program is changed
# to "Easel", the sample server program on the following page can be used
# in place of Excel as a server.


include esldde.inc          # DDE stimulus definitions

#################################
# Stimulus and Variable definitions

stimulus DDE "esldde"               # DDE stimulus declaration

integer variable  ConvID            # Conversation ID, supplied by DDE DLL
                  Status            # Conversation status flags
                  ErrorID           # ID of last error which occurred

boolean variable  LocalTermination is false     # Which partner terminates
                                                # first

string variable  AppName is "Excel" # Partner application name, change this
                                    # to "Easel" to connect to the sample
                                    # server program
             Topic is "sheet1.xls"        # Conversation topic
             ItemName is "r1"             # Item name for data exchange
             Format is DDEFMT_TEXT        # Format of exchanged data


#################################
# Primary Region Definition

enabled visible color 26 primary textual region PrimaryWindow
    size 500 200 at position 8 50
    black foreground
    title bar "Sample Easel DDE Client"
    border
    system menu
    minimize button
    maximize button
```

```
##################################
# Response Definitions

# Attempt to start a conversation as a client with the server (Excel),
# which must be running, on the topic "sheet1.xls". If successful, request
# a copy of row 1 (ItemName = "r1")

response to start
    copy DDEInitiate (AppName, Topic) to ConvID
    if (ConvID = 0) then                # Excel did not respond, or cannot
       add to PrimaryWindow             # converse on the specified topic
          insert "ERROR: No response from " AppName " on " Topic "\n"
    else
       add to PrimaryWindow insert
          "Connected..Requesting.."
              call DDERequest (ConvID, ItemName, Format, Status)
    end if

response to stimulus DDE DDE_DATA       # The data has arrived
    add to PrimaryWindow insert
       "\nDATA: " DDEGetStringData (ConvID) "\n"
    copy true to LocalTermination
    call DDETerminate (ConvID)

response to stimulus DDE DDE_NACK       # Partner cannot supply the data
    add to PrimaryWindow insert
       "\nERROR: " ItemName "not available;terminating conversation\n"
    copy true to LocalTermination
    call DDETerminate (ConvID)


# For a conversation termination event, check to see if the termination was
# initiated locally, or by the partner. If partner generated, acknowledge,
# (although the DDE DLL would do this by default anyway). Otherwise this is
# just the partner acknowledging the LOCAL request for termination.

response to stimulus DDE DDE_TERMINATE
    if (LocalTermination = false) then
       call DDETerminate (ConvID)
    end if
    add to PrimaryWindow insert
       "Conversation terminated\n"

# Simply report errors to the screen, and terminate.

response to stimulus DDE DDE_ERROR
    copy DDEGetErrorID (ConvID) to ErrorID
    add to PrimaryWindow insert
       "ERROR: " DDEGetErrorString (ConvID) "; terminating conversation\n"

    copy true to LocalTermination
    call DDETerminate (ConvID)

# When the user closes the application, make sure all conversations are
# terminated.

response to PrimaryWindow
    on close
       call DDETerminateAll ()
       exit
```

1.3.3.2 Server Program

```
########################################################################
# This program is a sample DDE server that illustrates the simplest
# server processing possible. It accepts a single DDE conversation on
# the application/topic pair "Easel"/"sheet1.xls". The only item name
# for which it accepts requests is "r1".
#
# Note that this program can be used as a server partner for the sample
# client program on the previous page if the application names are made
# to match.

##################################
# Stimulus and Variable declarations

include eslddе.inc                  # DDE library definitions
stimulus DDE "eslddе"               # DDE stimulus declaration

integer variable  ConvID is 0       # Conversation ID, supplied by DDE DLL
                  TempID            # for terminating extra conversations.
                  Status            # Conversation status flags
                  ErrorID           # ID of last error that occurred
                  DataLength        # For DDEDataString call

boolean variable  WildCard is true        # For DDERegisterServer

string variable  AppName is "Easel"       # Local (server) application name
                 Topic is "sheet1.xls"    # Conversation topic
                 ItemName                 # Item name for data exchange
                 Format                   # Format requested by client
                 DataString               # Buffer for data

##################################
# Primary Region Definition

enabled visible color 26 primary textual region PrimaryWindow
    size 500 200 at position 8 50
    black foreground
    title bar "Sample Easel DDE Server"
    border
    system menu
    minimize button
    maximize button
```

```
##################################
# Response Definitions

# Register one application/topic name pair, and then wait for a client
# to initiate a conversation.

response to start
    call DDERegisterServer (AppName, Topic, WildCard)
    add to PrimaryWindow insert "Waiting..."

# A client has asked for a conversation, and the DDE DLL has accepted on
# our behalf.  Once the first conversation is established, this program
# terminates subsequent client initiations, as this program only handles a
# single conversation.  (Note that this does not guarantee that the client
# did not issue any requests; if a request IS received from a partner that
# has been terminated, it should be ignored.)

response to stimulus DDE DDE_INITIATE
    if (ConvID = 0) then                    # First conversation
        copy eventparam to ConvID
        add to PrimaryWindow insert
           "Initiate received, accepted\n"
    else                                    # Accept only 1 conversation
        add to PrimaryWindow insert
           "\nanother initiate received, terminated\n"
            copy eventparam to TempID
            call DDETerminate (TempID)
    end if

# A client is requesting data; ignore requests from partners we have
# terminated.  Otherwise, make sure we recognize the ITEM requested AND the
# format.  If so, call DDEDataString to send the data.  If not, allow the
# DDE DLL to perform its default processing, which is a negative
# acknowledgement.

response to stimulus DDE DDE_REQUEST
    if (eventparam = ConvID) then               # Ignore outdated requests
       copy DDEGetItemName (ConvID) to ItemName
       if (ItemName = "r1") then                # Check the item name
           copy DDEGetFormat (ConvID) to Format
           if (Format = DDEFMT_TEXT) then       # Check the format
             copy "10 20 30\n" to DataString
             copy 0 to Status
             copy (length of DataString) to DataLength
             add to PrimaryWindow insert "request received, data sent\n"
             call DDEDataString (ConvID, ItemName, DataString,
                   DataLength,Format, Status)
           end if
       end if
    end if
```

```
# If the main conversation is being terminated, report it.  Otherwise, this
# termination event is an acknowledgement for a terminate sent in the
# response to stimulus DDE DDE_INITIATE above.

response to stimulus DDE DDE_TERMINATE
    if (eventparam = ConvID) then     # Main conversation is being term'd
       add to PrimaryWindow insert "Conversation terminated by partner\n"
          copy 0 to ConvID
    end if

# Simply report errors to the screen, and terminate.

response to stimulus DDE DDE_ERROR
    copy DDEGetErrorID (ConvID) to ErrorID
    add to PrimaryWindow insert
        "ERROR: " DDEGetErrorString (ConvID) "; terminating conversation\n"
    call DDETerminate (ConvID)

# When the user closes the application, make sure all conversations are
# terminated.

response to PrimaryWindow
    on close
        call DDETerminateAll ()
        exit
```

1.4 The DDE Functions and Subroutines

Many of the DDE functions and subroutines require the ID of an established conversation as an argument.

DDE Participant	Typically Obtains ID
Client	As a return value from **DDEInitiate**.
Server	From the built-in function **eventparam** during execution of a stimulus response.

1.4.1 Summary of Functions and Subroutines

The following table summarizes the DDE functions and subroutines. Following the summary, each function and subroutine is discussed in detail.

Function/Subroutine	See Page	Description
Polling for Topics		
DDEPollTopics	1-19	Called by a client to poll any currently running servers for the application/topic name pairs they are serving.
DDEEnumeratePollTopics	1-20	Called by a client to enumerate all entries in the topics poll list that result from the most recent call to **DDEPollTopics**.
DDEGetPollAppName	1-21	Called by a client while looping through the poll list created by **DDEPollTopics** to determine the application name that resulted from a poll.
DDEGetPollTopic	1-22	Called by a client while looping through the poll list created by **DDEPollTopics** to determine the topic name that resulted from a poll.

Function/Subroutine	See Page	Description
Starting Conversations		
DDEInitiate	1-24	Called by a client. Attempts to initiate a single conversation with an application/topic pair.
DDERegisterServer	1-25	Called by a server to register an application/topic name pair with the DDE DLL.
DDEUnRegisterServer	1-27	Called by a server to remove an application/topic pair from the list of pairs for which the caller is willing to serve.
DDEEnumerateConversations	1-28	Called by a client or server to get the conversation ID of each active conversation.
Exchanging Data		
DDERequest	1-30	Called by a client to make a one-time data request.
DDEDataString	1-31	Called by a server to send data to the client.
DDEExecute	1-33	Called by a client to request that the server execute one or more commands.
DDEAdvise	1-34	Called by a client to request that an advise state be established.
DDEUnAdvise	1-36	Called by a client to request that an advise state be terminated.
DDEPokeString	1-37	Called by a client to send unsolicited data to the server.
DDEAck	1-39	Called by a client or server to issue an acknowledgment.
Terminating Conversations		
DDETerminate	1-41	Called by a client or server to terminate a conversation.
DDETerminateAll	1-42	Called by a client or server to terminate all active conversations.

Function/Subroutine	See Page	Description
Getting Information on Existing Conversations		
DDEGetAppName	1-45	Called by a client or server. Returns a string that is the application name associated with the conversation specified by the ConvID argument.
DDEGetTopic	1-46	Called by a client or server. Returns a string that is the topic name associated with the conversation specified by the ConvID argument.
DDEGetFormat	1-47	Called by a client or server. Returns a string that is the format name associated with an event for the conversation specified by the ConvID argument.
DDEGetItemName	1-48	Called by a client or server. Returns a string that is the item name associated with an event for the conversation specified by the ConvID argument.
DDEGetStringData	1-49	Called by a client or server. Returns a string that is the text data associated with an event for the conversation specified by the ConvID argument.
DDEGetStatus	1-50	Called by a client or server. Returns a string that is the status value associated with an event for the conversation specified by the ConvID argument.
DDEGetErrorID	1-51	Called by a client or server. Returns the ID of the most recent error encountered for a given conversation.

Function/Subroutine	See Page	Description
Manipulating Status Variables		
DDESetStatusFlag	1-53	Called by a client or server. Used to compose status parameters for those function and subroutine calls that require them.
DDEQueryStatusFlag	1-54	Called by a client or server. Used to decompose status values returned by the DDE function **DDEGetStatus**.
Miscellaneous Calls		
DDERegisterFormat	1-56	Called by a client or server. Registers a format for use in a DDE exchange.
DDEGetPID	1-57	Called by a server. Returns the OS/2 process ID associated with the current instance of EASEL OS/2 EE.
DDEGetErrorString	1-58	Called by a client or server. Accepts an error identifier (obtained from a call to **DDEGetErrorID**), and returns the error string associated with that ID.
DDESetTraceLevel	1-59	Called by a client or server. Sets the trace level so that you can see a trace of the DDE messages and/or the DDE DLL's actions when you are developing DDE applications.

1.4.2 Polling For Topics

Sometimes it is useful to test to see what server application/topic pairs are available at a given time. This section describes four functions that you can use to obtain this information.

Function	See Page	Description
DDEPollTopics	1-19	Called by a client to poll any currently running servers for the application/topic name pairs they are serving.
DDEEnumeratePollTopics	1-20	Called by a client to enumerate all entries in the topics poll list that result from the most recent call to **DDEPollTopics**.
DDEGetPollAppName	1-21	Called by a client while looping through the poll list created by **DDEPollTopics** to determine the application name that resulted from a poll.
DDEGetPollTopic	1-22	Called by a client while looping through the poll list created by **DDEPollTopics** to determine the topic name that resulted from a poll.

1.4.2.1 DDEPollTopics Function

Called by: Client

Action: Polls any currently running servers for the application/topic name pairs they are serving.

PM Message: Broadcasts a WM_DDE_INITIATE message.

Declaration:
```
function DDEPollTopics (string:AppName, string:Topic)
                  returns integer library "esldde"
```

Where: AppName is the server's DDE application name. (If the server is an EASEL OS/2 EE program, it is the registered name.)

Topic is the server's associated DDE topic name. (If the server is an EASEL OS/2 EE program, it is the registered name.)

Remarks: This function attempts to initiate conversations with all active servers, recording the application/topic pairs supplied by each responding server. (See 1.4.3.2, "DDERegisterServer Subroutine" on page 1-25 for the DDE protocol rules relating to how the server responds to client requests.) Any conversation initiated as a result of this poll is immediately terminated. Therefore, no permanent conversations are started as a result of this call. You can examine the list that results from this poll by calling **DDEEnumeratePollTopics** along with **DDEGetPollAppName** and **DDEGetPollTopic**. The return value is the number of entries in the list.

Poll topics do not accumulate across calls; each call to this function clears the previous list before the new entries are recorded.

Example:
```
# Find all servers with "Easel" application name,
# and display their topics

copy 0 to LastTopic
copy DDEPollTopics ("Easel", "") to NumConvs
while (NumConvs > 0) loop
   copy DDEEnumeratePollTopics (LastTopic) to LastTopic
   add to TopicWindow insert at cursor
       DDEGetPollTopic (LastTopic) "\n"
   copy (NumConvs - 1) to NumConvs
end loop
```

1.4.2.2 DDEEnumeratePollTopics Function

Called by: Client

Action: Enumerates all entries in the topics poll list that result from the most recent call to **DDEPollTopics**.

Declaration:
```
function DDEEnumeratePollTopics (integer:LastTopic)
                    returns integer library "esldde"
```

Where: LastTopic is 0 to start, and then is the return value from the previous call.

Remarks: To begin enumeration of the current list, use an argument of 0. Each subsequent call should pass the value returned by the previous call as the LastTopic argument. When all entries have been enumerated, the return value will be 0. (You can use the value returned as an argument to **DDEGetPollAppName** and **DDEGetPollTopic**. Do not use the value returned as an argument to any other DDE function or subroutine except **DDEEnumeratePollTopics**).

Do not call **DDEPollTopics** while enumerating poll topics, as the results will be unpredictable.

Example:
```
# Display the application names and topics for
# all servers

copy 0 to LastTopic
copy DDEPollTopics ("", "") to NumConvs
while (NumConvs > 0) loop
   copy DDEEnumeratePollTopics (LastTopic) to LastTopic
   add to TopicWindow insert at cursor
       DDEGetPollAppName (LastTopic) " "
       DDEGetPollTopic (LastTopic) "\n"
   copy (NumConvs - 1) to NumConvs
end loop
```

1.4.2.3 DDEGetPollAppName Function

Called by: Client

Action: You can use this function while looping through the poll list created by **DDEPollTopics** to determine the application name that resulted from a poll.

Declaration:

```
function DDEGetPollAppName (integer:LastTopic)
                    returns string library "esldde"
```

Where: LastTopic is a value returned from a call to **DDEEnumeratePollTopics** within the current response.

Example:

```
# Display the application names of all servers who know
# about topic "mytopic"

copy 0 to LastTopic
copy DDEPollTopics ("", "mytopic") to NumConvs
while (NumConvs > 0) loop
   copy DDEEnumeratePollTopics (LastTopic) to LastTopic
   add to TopicWindow insert at cursor
       DDEGetPollAppName (LastTopic) "\n"
   copy (NumConvs - 1) to NumConvs
end loop
```

1.4.2.4 DDEGetPollTopic Function

Called by: Client

Action: You can use this function while looping through the poll list created by **DDEPollTopics** to determine the topic name that resulted from a poll.

Declaration:
```
function DDEGetPollTopic (integer:LastTopic)
                returns string library "esldde"
```

Where: LastTopic is a value returned from a call to **DDEEnumeratePollTopics** within the current response.

Example:
```
# Display the topic names of all servers with
# application "Easel"

copy 0 to LastTopic
copy DDEPollTopics ("Easel", "") to NumConvs
while (NumConvs > 0) loop
   copy DDEEnumeratePollTopics (LastTopic) to LastTopic
   add to TopicWindow insert at cursor
       DDEGetPollTopic (LastTopic) "\n"
   copy (NumConvs - 1) to NumConvs
end loop
```

1.4.3 Starting Conversations

This section describes the functions and subroutines necessary for both servers and clients to start conversations, as well as functions used for keeping track of active conversations.

Function/Subroutine	See Page	Description
DDEInitiate	1-24	Called by a client. Attempts to initiate a single conversation with an application/topic pair.
DDERegisterServer	1-25	Called by a server to register an application/topic name pair with the DDE DLL.
DDEUnRegisterServer	1-27	Called by a server to remove an application/topic pair from the list of pairs for which the caller is willing to serve.
DDEEnumerateConversations	1-28	Called by a client or server to get the conversation ID of each active conversation.

1.4.3.1 DDEInitiate Function

Called by: Client

Action: Attempts to initiate a single conversation on a specific application/topic pair.

PM Message: Broadcasts a WM_DDE_INITIATE message.

Sends a WM_DDE_INITIATEACK message to the first respondent, and sends a WM_DDE_TERMINATE message to all servers who subsequently respond to this broadcast.

Declaration:

```
function DDEInitiate (string:AppName, string:Topic)
                  returns integer library "esldde"
```

Where: AppName is the server's DDE application name. (If the server is an EASEL OS/2 EE program, it is the registered name.) It cannot be a zero-length string.

Topic is the server's associated DDE topic name. (If the server is an EASEL OS/2 EE program, it is the registered name.) It cannot be a zero-length string.

Remarks: AppName and Topic may have been obtained from a previous poll. Note that this does not guarantee that a conversation will be started.

This function returns a conversation ID if it has successfully acknowledged a conversation, or 0 otherwise.

Example:

```
response to InitiateKey
copy "Easel" to AppName
copy "MyTopic" to Topic
copy DDEInitiate (AppName, Topic) to ConvID
if (ConvID = 0) then
   # no conversation established
else
   make ConnectKey green
end if
```

1.4.3.2 DDERegisterServer Subroutine

Called by: Server

Action: Registers an application/topic name pair with the DDE DLL.

Declaration:

```
subroutine DDERegisterServer (string:AppName,
                              string:Topic,
                              boolean:WildCard)
                      library "esldde"
```

Where: AppName is the DDE application name by which the server is to be known. This cannot be a zero-length string.

Topic is the associated DDE topic name of the server. This cannot be a zero-length string.

WildCard is a boolean. When set to **true**, the server will respond to all client requests that match based on the DDE protocol rules. (See DDE Protocol Rules on page 1-25.) When set to **false**, the server will respond only to exact matches. (See WildCard on page 1-26.)

Remarks: Once this subroutine is called, the DDE DLL responds to all DDE conversation initiation attempts (according to the DDE protocol rules outlined below) by

- Sending the WM_DDE_INITIATEACK message to the partner (thereby accepting the conversation), and
- Invoking a DDE_INITIATE event in the EASEL OS/2 EE program.

DDE Protocol Rules: The DDE protocol prescribes that a server should respond to client initiation requests according to the following rules:

Application Name	Topic Name	Server Responds Once
zero length	zero length	For each application/topic pair it supports.
zero length	specified	If it supports that topic.
specified	zero length	For each specified topic it supports, if the application name matches the servers.
specified	specified	If it supports that topic under that application name.

The DDE DLL keeps a list of all application/topic name pairs registered by the EASEL OS/2 EE server application, and follows the rules outlined above, based on the list in effect at the time a WM_DDE_INITIATE message is received.

WildCard: The DDE DLL will *not* respond based on the first three rules above for any application/topic pair registered with WildCard set to **false**. This is useful for servers that want to respond to requests from clients that specify a particular *instance* of that server.

Many DDE-compliant applications use their application name, appended with the OS/2 process ID (which is available to an EASEL OS/2 EE program through **DDEGetPID**, described in 1.4.8, "Miscellaneous Calls" on page 1-55) for this purpose. Such a server would probably register each topic that it supports under its regular name, with WildCard set to **true**, and then register each topic again with the instance-specific name, with WildCard set to **false**. This would allow the server to respond to requests from clients that know the process ID of the *instance* of the application with which they want to converse.

Example:

```
response to start
   copy "EaselApp" to AppName
   copy "System" to Topic
   copy true to WildCard
   call DDERegisterServer (AppName, Topic, WildCard)
   copy "Hotlink" to Topic
   call DDERegisterServer (AppName, Topic, WildCard)
```

1.4.3.3 DDEUnRegisterServer Subroutine

Called by: Server

Action: Removes an application/topic pair from the list of pairs for which the caller is willing to serve.

Declaration:
```
subroutine DDEUnRegisterServer (string:AppName,
                                string:Topic)
                        library "esldde"
```

Where: AppName is the DDE application name passed to **DDERegisterServer**.

Topic is the topic passed to **DDERegisterServer**.

Remarks: This subroutine has no effect on existing, active conversations. Once an application/topic pair is removed from the list, the DDE DLL will no longer accept conversations that specify that pair.

Example:
```
response to Unregister
   copy "EaselApp" to AppName
   copy "Hotlink" to Topic
   call DDEUnRegisterServer (AppName, Topic)
```

1.4.3.4 DDEEnumerateConversations Function

Called by: Client or Server

Action: You can call this function repeatedly to get the conversation ID of each active conversation.

Declaration:
```
function DDEEnumerateConversations (integer:ConvID)
                returns integer library "esldde"
```

Where: ConvID should be 0 to begin the enumeration. Each subsequent call should pass the conversation ID obtained from the return value of the previous call. When all active conversations have been enumerated, the function will return a conversation ID of 0.

Example:
```
response to Enumerate
   copy 0 to LastConv
   copy PermIndex to NumConvs
   while (NumConvs > 0) loop
      copy DDEEnumerateConversations (LastConv) to LastCo
      add to ConvWindow insert at cursor
         DDEGetAppName (LastConv) " "
         DDEGetTopic (LastConv) " "
         LastConv "\n"
      copy (NumConvs - 1) to NumConvs
   end loop
```

1.4.4 Exchanging Data

This section describes the subroutines used to carry out the six types of data exchange transactions, as well as the subroutine for acknowledging receipt of a data exchange request.

Subroutine	See Page	Description
DDERequest	1-30	Called by a client to make a one-time data request.
DDEDataString	1-31	Called by a server to send data to the client.
DDEExecute	1-33	Called by a client to request that the server execute one or more commands.
DDEAdvise	1-34	Called by a client to request that an advise state be established.
DDEUnAdvise	1-36	Called by a client to request that an advise state be terminated.
DDEPokeString	1-37	Called by a client to send unsolicited data to the server.
DDEAck	1-39	Called by a client or server to issue an acknowledgment.

1.4.4.1 DDERequest Subroutine

Called by: Client

Action: Makes a one-time data request.

PM Message: Posts a WM_DDE_REQUEST message to the server partner.

Declaration:

```
subroutine DDERequest (integer:ConvID, string:ItemName,
                       string:Format, integer:Status)
                      library "esldde"
```

Where: ConvID is the conversation ID assigned by the DDE DLL.

ItemName is a non-zero length string specifying the item of interest, which should have some meaning to the server.

Format is either DDEFMT_TEXT (defined in ESLDDE.INC), or any format string that has been registered through a call to **DDERegisterFormat**.

Status must be 0.

Required Partner Response: Upon receipt of a one-time data request, the DDE protocol requires that the server respond as follows. The client should not attempt another DDE exchange until the server has responded with one of these events.

Acknowledgment	EASEL OS/2 EE Event Invoked	PM Message
Positive (Request can be honored)	DDE_DATA	WM_DDE_DATA
Negative (Request cannot be honored)	DDE_NACK	WM_DDE_ACK with DDE_FACK status bit cleared

Example:

```
response to RequestKey
   # Get ConvID
   if (not (ConvID = 0)) then  # make sure a conversation
                               # is active
      copy "itemname" to ItemName
      copy DDEFMT_TEXT to Format
      copy 0 to Status
      call DDERequest (ConvID, ItemName, Format, Status)
   end if
```

1.4.4.2 DDEDataString Subroutine

Called by: Server

Action: Sends data to the client.

PM Message: Posts a WM_DDE_DATA message to the client partner.

Declaration:

```
subroutine DDEDataString (integer:ConvID,
                          string:ItemName,
                          string:DataString,
                          integer:DataLength,
                          string:Format, integer:Status)
                library "esldde"
```

Where: ConvID is the conversation ID assigned by the DDE DLL.

ItemName should contain the same name that was requested by the client.

DataString is the data string being sent to the client. It can be zero length, although normally only for data sent as part of an ongoing advise state.

DataLength is the length of the data string.

Format should be the same format requested by the client.

Status is composed of DDE_FACKREQ and DDE_FRESPONSE status flags. (See Status Flags below.)

Remarks: This subroutine should be called only in response to a DDE_REQUEST event, or as part of an ongoing DDE_ADVISE.

Status Flags: The DDE protocol defines the DDE_FACKREQ and DDE_FRESPONSE status flags for the WM_DDE_DATA message, which is posted by this subroutine, as follows:

- The DDE_FRESPONSE status flag should be set if this data is being sent as part of an ongoing advise state.
- The DDE_FACKREQ status flag should be set if the application wants the receiving partner to acknowledge receipt of the data (the DDE DLL will invoke a DDE_ACK event when the partner does this). If this data is being sent as part of an ongoing advise, this flag should be set if the client requested that it be set when the advise state was established. (See DDE_ADVISE in 1.5.3, "DDE Events List" on page 1-63.)

Required Partner Response: Upon receipt of data, the DDE protocol requires that the client respond as follows, if the DDE_FACKREQ status flag is set. The server should not attempt another DDE exchange until the client has responded with one of these events.

Acknowledgment	**EASEL OS/2 EE Event Invoked**	**PM Message**
Positive (Data successfully received)	DDE_ACK	WM_DDE_ACK with DDE_FACK status bit set
Negative (Data not successfully received)	DDE_NACK	WM_DDE_ACK with DDE_FACK status bit cleared

Example:

```
response to stimulus DDE DDE_REQUEST
   copy eventparam to ConvID
   copy DDEGetItemName (ConvID) to ItemName
   copy DDEGetFormat (ConvID) to Format
   if (GetTheData (ItemName, Format, DataString) =
     true) then
       copy (length of DataString) to DataLength
       copy DDESetStatusFlag (0, DDE_FACKREQ) to Status
       call DDEDataString (ConvID, ItemName, DataString,
                           DataLength, Format, Status)
   else
       copy 0 to Status              # Send a negative ACK
       call DDEAck (ConvID, ItemName, Format, Status)
   end if

response to stimulus DDE DDE_ACK     # Recognize the
   copy true to StatusOK             # acknowledgment
```

1.4.4.3 DDEExecute Subroutine

Called by: Client

Action: Requests that the server execute one or more commands.

PM Message: Posts a WM_DDE_EXECUTE message to the server partner.

Declaration:

```
subroutine DDEExecute (integer:ConvID, string:Command,
                       integer:DataLength,
                       integer:Status)
                      library "esldde"
```

Where: ConvID is the conversation ID assigned by the DDE DLL.

Command is the command that the server is to execute. The format of these commands is not defined by the DDE protocol, although the DDE protocol definition document does suggest a format. Consult the documentation for the particular server for which the commands are intended.

DataLength is the length of the command.

Status must be 0.

Required Partner Response: Upon receipt of a request that the server execute one or more commands, the DDE protocol requires that the server respond as follows. The client should not attempt another DDE exchange until the server has responded with one of these events.

Acknowledgment	**EASEL OS/2 EE Event Invoked**	**PM Message**
Positive (Command successfully executed)	DDE_ACK	WM_DDE_ACK with DDE_FACK status bit set
Negative (Command not successfully executed)	DDE_NACK	WM_DDE_ACK with DDE_FACK status bit cleared

Example:

```
response to ExecuteKey
   # Get ConvID
   copy "commands to execute" to Command
   copy (length of Command) to DataLength
   copy 0 to Status
   call DDEExecute (ConvID, Command, DataLength, Status)
```

1.4.4.4 DDEAdvise Subroutine

Called by: Client

Action: Requests that an advise state be established.

PM Message: Posts a WM_DDE_ADVISE message to the server partner.

Declaration:

```
subroutine DDEAdvise (integer:ConvID, string:ItemName,
                      string:Format, integer:Status)
                      library "esldde"
```

Where: ConvID is the conversation ID assigned by the DDE DLL.

ItemName is a non-zero-length string that describes the item of interest.

Format is either DDEFMT_TEXT, or some other format string that has been registered by **DDERegisterFormat**.

Status is composed only of DDE_FACKREQ and DDE_FNODATA.

Status Flags: The DDE protocol defines the use of the DDE_FACKREQ and DDE_FNODATA flags for the WM_DDE_ADVISE message, which is posted by this subroutine, as follows:

- If DDE_FACKREQ is set, the client is requesting that the server send subsequent WM_DDE_DATA messages for this advise with the DDE_FACKREQ flag set.
- If DDE_FNODATA is set, the client is requesting that the server send subsequent WM_DDE_DATA messages for this advise with a zero-length data portion. The client might request this when the cost associated with actually rendering the data may be high. If this flag is set, the server will simply inform the client that data is available. If, upon receipt of this event (DDE_DATA event with a zero-length data portion), the client wants the data, it can request it by calling **DDERequest**. The client should remember not to attempt to retrieve data with **DDEGetStringData** if advise was established with the DDE_FNODATA flag set.

Required Partner Response: Upon receipt of a request that an advise state be established, the DDE protocol requires that the server respond as follows. The client should not attempt another DDE exchange until the server has responded with one of these events.

Acknowledgment	EASEL OS/2 EE Event Invoked	PM Message
Positive (Advise state established)	DDE_ACK	WM_DDE_ACK with DDE_FACK status bit set
Negative (Advise state not established)	DDE_NACK	WM_DDE_ACK with DDE_FACK status bit cleared

Example:

```
response to HotLinkKey
   # Get ConvID
   copy "itemname" to ItemName
   copy DDEFMT_TEXT to Format
   copy DDESetStatusFlag (0, DDE_FNODATA) to Status
   call DDEAdvise (ConvID, ItemName, Format, Status)

response to stimulus DDE DDE_ACK
   copy true to HotLinkOK

response to stimulus DDE DDE_NACK
   copy false to HotLinkOK
```

1.4.4.5 DDEUnAdvise Subroutine

Called by: Client

Action: Requests that an advise state be terminated.

PM Message: Posts a WM_DDE_UNADVISE message to the server partner.

Declaration:
```
subroutine DDEUnAdvise (integer:ConvID, string:ItemName,
                        string:Format, integer:Status)
                       library "esldde"
```

Where: ConvID is the conversation ID assigned by the DDE DLL.

ItemName is a string that describes the item of interest. If this string is zero length, the server will terminate *all* advise states associated with this conversation ID.

Format is either DDEFMT_TEXT or any other format string that has been registered by **DDERegisterFormat**.

Status must be 0.

Required Partner Response: Upon receipt of a request that an advise state be terminated, the DDE protocol requires that the server respond as follows. The client should not attempt another DDE exchange until the server has responded with one of these events.

Acknowledgment	EASEL OS/2 EE Event Invoked	PM Message
Positive (Advise state terminated)	DDE_ACK	WM_DDE_ACK with DDE_FACK status bit set
Negative (Advise state not terminated)	DDE_NACK	WM_DDE_ACK with DDE_FACK status bit cleared

Example:
```
response to NewTopic
   # Get ConvID
   copy "itemname" to ItemName
   copy DDEFMT_TEXT to Format
   copy 0 to Status
   call DDEUnAdvise (ConvID, ItemName, Format, Status)
```

1.4.4.6 DDEPokeString Subroutine

Called by: Client

Action: Sends unsolicited data to the server.

Remarks: Posts a WM_DDE_POKE message to the server partner.

Declaration:

```
subroutine DDEPokeString (integer:ConvID,
                          string:ItemName,
                          string:DataString
                          integer:DataLength,
                          string:Format, integer:Status)
                  library "esldde"
```

Where: ConvID is the conversation ID assigned by the DDE DLL.

ItemName is a non-zero-length string that describes the item of interest.

DataString is the data string.

DataLength is the data length.

Format is either DDEFMT_TEXT, or some other format string that has been registered by **DDERegisterFormat**.

Status is 0.

Required Partner Response: Upon receiving unsolicited data, the DDE protocol requires that the server respond as follows. The client should not attempt another DDE exchange until the server has responded with one of these events.

Acknowledgment	**EASEL OS/2 EE Event Invoked**	**PM Message**
Positive (Command successfully executed)	DDE_ACK	WM_DDE_ACK with DDE_FACK status bit set
Negative (Command not successfully executed)	DDE_NACK	WM_DDE_ACK with DDE_FACK status bit cleared

Example:

```
response to PokeKey
   # Get ConvID
   copy "itemname" to ItemName
   copy "poke data" to DataString
   copy (length of DataString) to DataLength
   copy DDEFMT_TEXT to Format
   copy 0 to Status
   call DDEPokeString (ConvID, ItemName, DataString,
                       DataLength, Format, Status)
```

1.4.4.7 DDEAck Subroutine

Called by: Client or Server

Action: Issues an acknowledgment.

PM Message: Posts a WM_DDE_ACK message to the conversation partner.

Declaration:

```
subroutine DDEAck (integer:ConvID, string:ItemName,
                   string:Format, integer:Status)
                      library "esldde"
```

Where: ConvID is the conversation ID assigned by the DDE DLL.

ItemName is a non-zero-length string that describes the item that is being acknowledged.

Format is the format of the data being acknowledged.

Status is one of the following status flag values (see 1.4.7, "Manipulating Status Variables" on page 1-52 for the meanings of these values):

DDE_FACK
DDE_FBUSY
DDE_FNOTPROCESSED

Availability: This subroutine should only be called in response to the following events:

DDE_POKE
DDE_EXECUTE
DDE_ADVISE
DDE_UNADVISE
DDE_DATA event with the DDE_FACKREQ status flag set
DDE_REQUEST

Example:

```
response to stimulus DDE DDE_EXECUTE
   copy eventparam to ConvID
   copy DDEGetStringData (ConvID) to Command
   copy "" to ItemName
   copy DDEGetFormat (ConvID) to Format
   if (CommandProcessor (Command) = true) then
      # Send a positive ACK
      copy DDESetStatusFlag (0, DDE_FACK) to Status
      call DDEAck (ConvID, ItemName, Format, Status)
   else
      copy 0 to Status      # Send a negative ACK
      call DDEAck (ConvID, ItemName, Format, Status)
   end if
```

1.4.5 Terminating Conversations

This section describes the subroutines used to terminate existing conversations.

Subroutine	See Page	Description
DDETerminate	1-41	Called by a client or server to terminate a conversation.
DDETerminateAll	1-42	Called by a client or server to terminate all active conversations.

1.4.5.1 DDETerminate Subroutine

Called by: Client or Server

Action: Terminates a conversation.

PM Message: Posts a WM_DDE_TERMINATE message to the conversation partner.

Declaration:
```
subroutine DDETerminate (integer:ConvID)
                         library "esldde"
```

Where: ConvID is the conversation ID assigned by the DDE DLL.

Remarks: The DDE protocol allows either partner to terminate a conversation at any time.

You can use this subroutine either

- To voluntarily terminate a conversation, or
- To respond to a partner-initiated termination.

The application that *receives* the termination event is required to respond by posting an acknowledging terminate event. No other responses are acceptable.

If you call this subroutine to voluntarily terminate a conversation, the DDE DLL will, at some time in the future, invoke a DDE_TERMINATE event when the partner acknowledges the termination.

Example:
```
response to PrimaryWindow
   # Get ConvID
   on close
      call DDETerminate (ConvID)
```

1.4.5.2 DDETerminateAll Subroutine

Called by: Client or Server

Action: Terminates all active conversations.

PM Message: Posts a WM_DDE_TERMINATE message to all active conversation partners, and returns immediately (does not wait for all partners to send the corresponding WM_DDE_TERMINATE message).

Declaration:

```
subroutine DDETerminateAll ()
                          library "esldde"
```

Example:

```
response to PrimaryWindow
   on close
    call DDETerminateAll ()
```

1.4.6 Getting Information on Existing Conversations

The DDE DLL maintains information about all active conversations, which is available to the EASEL OS/2 EE program through a variety of functions. This section describes those functions.

Only the application name, topic name, and most recent error ID are stored with the conversation for the entire life of the conversation. Other information, such as the current item name, format, or status flags, should only be queried while executing a response to a DDE event for which that information is relevant. The DDE events listed in 1.5, "The DDE Events" on page 1-60 describe which information is available during each event.

Function	See Page	Description
DDEGetAppName	1-45	Called by a client or server. Returns a string that is the application name associated with the conversation specified by the ConvID argument.
DDEGetTopic	1-46	Called by a client or server. Returns a string that is the topic name associated with the conversation specified by the ConvID argument.
DDEGetFormat	1-47	Called by a client or server. Returns a string that is the format name associated with an event for the conversation specified by the ConvID argument.
DDEGetItemName	1-48	Called by a client or server. Returns a string that is the item name associated with an event for the conversation specified by the ConvID argument.
DDEGetStringData	1-49	Called by a client or server. Returns a string that is the text data associated with an event for the conversation specified by the ConvID argument.

Function	See Page	Description
DDEGetStatus	1-50	Called by a client or server. Returns a string that is the status value associated with an event for the conversation specified by the ConvID argument.
DDEGetErrorID	1-51	Called by a client or server. Returns the ID of the most recent error encountered for a given conversation.

1.4.6.1 DDEGetAppName Function

Called by: Client or Server

Action: Returns a string that is the application name associated with the conversation specified by the ConvID argument.

Declaration:
```
function DDEGetAppName (integer:ConvID)
                returns string library "esldde"
```

Where: ConvID is the conversation ID assigned by the DDE DLL.

Availability: This function is available as long as the conversation is active.

Example:
```
response to Enumerate
   copy 0 to LastConv
   copy PermIndex to NumConvs
   while (NumConvs > 0) loop
      copy DDEEnumerateConversations (LastConv) to LastConv
      add to ConvWindow insert at cursor
          DDEGetAppName (LastConv) "\n"
      copy (NumConvs - 1) to NumConvs
   end loop
```

1.4.6.2 DDEGetTopic Function

Called by: Client or Server

Action: Returns a string that is the topic name associated with the conversation specified by the ConvID argument.

Declaration:
```
function DDEGetTopic (integer:ConvID)
                  returns string library "esldde"
```

Where: ConvID is the conversation ID assigned by the DDE DLL.

Availability: This function is available as long as the conversation is active.

Example:
```
response to Enumerate
   copy 0 to LastConv
   copy PermIndex to NumConvs
   while (NumConvs > 0) loop
      copy DDEEnumerateConversations (LastConv) to LastCc
      add to ConvWindow insert at cursor
         DDEGetTopic (LastConv) "\n"
      copy (NumConvs - 1) to NumConvs
   end loop
```

1.4.6.3 DDEGetFormat Function

Called by: Client or Server

Action: Returns a string that is the format name associated with the current event for the conversation specified by the ConvID argument.

Declaration:
```
function DDEGetFormat (integer: ConvID)
                    return string library "esldde"
```

Where: ConvID is the conversation ID assigned by the DDE DLL.

Availability: This function should only be called in response to the following events, to determine if the format is one that the application can handle, or to determine how to interpret the data:

DDE_REQUEST
DDE_DATA
DDE_ADVISE
DDE_UNADVISE
DDE_POKE
DDE_EXECUTE
DDE_ACK
DDE_NACK

Remarks: DDE predefines one text format, represented by the constant DDEFMT_TEXT (which is defined in esldde.inc as the string "**#1**"). The application should use this constant when testing the return value of **DDEGetFormat** to determine if the data is in plain text format, since many other applications may request other formats before accepting plain text format.

Example:
```
response to stimulus DDE DDE_ADVISE
   copy eventparam to ConvID
   copy DDEGetFormat (ConvID) to Format
   if (Format = DDEFMT_TEXT) then
       # Plain text format
       # Accept the advise state
   else
       # Reject this advise
   end if
```

1.4.6.4 DDEGetItemName Function

Called by: Client or Server

Action: Returns a string that is the item name associated with the current event for the conversation specified by the ConvID argument.

Declaration:
```
function DDEGetItemName (integer:ConvID)
                  returns string library "esldde"
```

Where: ConvID is the conversation ID assigned by the DDE DLL.

Availability: This function should only be called in response to the following events:

DDE_DATA
DDE_REQUEST
DDE_ADVISE
DDE_UNADVISE
DDE_POKE
DDE_ACK
DDE_NACK

Example:
```
response to stimulus DDE DDE_REQUEST
   copy eventparam to ConvID
   copy DDEGetItemName (ConvID) to ItemName
```

1.4.6.5 DDEGetStringData Function

Called by: Client or Server

Action: Returns a string that is the text data associated with the current event for the conversation specified by the ConvID argument.

Declaration:
```
function DDEGetStringData (integer:ConvID)
                  returns string library "esldde"
```

Where: ConvID is the conversation ID assigned by the DDE DLL.

Availability: This function should only be called in response to the following events:

DDE_DATA
DDE_EXECUTE
DDE_POKE

Example:
```
response to stimulus DDE DDE_DATA
   copy eventparam to ConvID
   copy DDEGetStringData (ConvID) to StringData
```

1.4.6.6 DDEGetStatus Function

Called by: Client or Server

Action: Returns a string that is the status associated with the current event for the conversation specified by the ConvID argument.

Declaration:
```
function DDEGetStatus (integer:ConvID)
                  returns integer library "esldde"
```

Where: ConvID is the conversation ID assigned by the DDE DLL.

Availability: This function should only be called in response to the following events:

DDE_DATA
DDE_REQUEST
DDE_ADVISE
DDE_UNADVISE
DDE_EXECUTE
DDE_POKE
DDE_ACK
DDE_NACK

Example:
```
response to stimulus DDE DDE_DATA
   copy eventparam to ConvID
   copy DDEGetStatus (ConvID) to Status
```

1.4.6.7 DDEGetErrorID Function

Called by: Client or Server

Action: Returns the ID of the most recent error encountered for a given conversation.

Declaration:
```
function DDEGetErrorID (integer:ConvID)
                returns integer library "esldde"
```

Remarks: The value returned by this function can be used as an argument to **DDEGetErrorString**.

Example:
```
response to stimulus DDE DDE_ERROR
   copy eventparam to ConvID
   copy DDEGetErrorID to ErrorID
   copy DDEGetErrorString (ErrorID) to ErrorString
```

1.4.7 Manipulating Status Variables

Many of the data exchange functions and subroutines accept a status argument, which is bundled by the DDE DLL with the PM message when it is delivered to the partner application. The DDE protocol prescribes the valid status values for each message, and what they mean. The EASEL OS/2 EE DDE DLL generates an error event if the status used for a given subroutine or function is not valid for the corresponding DDE message. When this happens, the message is not sent. (The description of each subroutine, function, and event in this document explains what status values are valid in each case, where applicable.)

A single status variable can encode multiple status flags. This section describes two functions for setting and querying the individual flags in a status variable. These can be used to construct an appropriate status parameter, or to decompose a status value received from a partner (through the **DDEGetStatus** call). The valid status flag values (defined in esldde.inc) are as follows:

Status Flag Value	Description
DDE_FACK	A positive acknowledgment.
DDE_FBUSY	Partner was busy. (Only valid if DDE_FACK flag is not set.)
DDE_FNODATA	Send subsequent DATA messages with 0 length data portion (for DDE_ADVISE).
DDE_FACKREQ	Request acknowledgment.
DDE_FRESPONSE	Data is sent in response to advise state.
DDE_FNOTPROCESSED	Data not processed. (Only valid if DDE_FACK flag is not set.)

Function	See Page	Description
DDESetStatusFlag	1-53	Called by a client or server. Used to compose status parameters for those function and subroutine calls that require them.
DDEQueryStatusFlag	1-54	Called by a client or server. Used to decompose status values returned by the function **DDEGetStatus**.

1.4.7.1 DDESetStatusFlag Function

Called by: Client or Server

Action: Used to compose Status parameters for those function and subroutine calls that require them.

Declaration:
```
function DDESetStatusFlag (integer:Status, integer:Flag)
                  returns integer library "esldde"
```

Where: Status is a DDE status value, which may already have encoded status flags.

Flag is a DDE status constant.

Remarks: This function accepts a DDE status value and a DDE status constant. It returns a new status value that encodes the Status constant into the old value.

Example:
```
response to Advise
   copy eventparam to ConvID
   copy DDEGetItemName (ConvID) to ItemName
   copy DDEGetFormat (ConvID) to Format
   copy DDESetStatusFlag (0, DDE_FACK) to Status
   call DDEAck (ConvID, ItemName, Format, Status) # send
                                              # a positive ACK
```

1.4.7.2 DDEQueryStatusFlag Function

Called by: Client or Server

Action: Used to decompose Status values returned by the DDE function **DDEGetStatus**.

Declaration:
```
function DDEQueryStatusFlag (integer:Status,
                             integer:Flag)
                returns boolean library "esldde"
```

Where: Status is a DDE status value.

Flag is a DDE status constant.

Remarks: This function accepts a DDE status value and a DDE status constant. It returns **true** if that status value has the status constant encoded within it, and **false** if it does not.

Example:
```
response to stimulus DDE DDE_ADVISE
   copy eventparam to ConvID
   copy DDEGetStatus (ConvID) to Status
   if (DDEQueryStatusFlag (Status, DDE_FNODATA) = true)
     then
     copy true to NoData
   else
     copy false to NoData
   end if
```

1.4.8 Miscellaneous Calls

Function/Subroutine	See Page	Description
DDERegisterFormat	1-56	Called by a client or server. Registers a format for use in a DDE exchange.
DDEGetPID	1-57	Called by a server. Returns the OS/2 process ID associated with the current instance of EASEL OS/2 EE.
DDEGetErrorString	1-58	Called by a client or server. Accepts an error identifier (obtained from a call to **DDEGetErrorID**), and returns the error string associated with that ID.
DDESetTraceLevel	1-59	Called by a client or server. Sets the trace level so that you can see a trace of the DDE messages and/or the DDE DLL's actions when you are developing DDE applications.

1.4.8.1 DDERegisterFormat Subroutine

Called by: Client or Server.

Action: Registers a format for use in a DDE exchange.

Declaration:
```
subroutine DDERegisterFormat (string:Format)
                              library "esldde"
```

Where: Format is the data format you want to register.

Remarks: An EASEL OS/2 EE program can only deal with text-based data.

The constant DDEFMT_TEXT (text format) is automatically registered for all DDE participants. Any other text-based data format referenced by a DDE partner must be registered in a system-wide table before it is used in a DDE exchange. This subroutine allows an EASEL OS/2 EE program to register such a text- based format. (Note that any format string returned from **DDEGetFormat** on an existing conversation can be assumed already to be registered, since, if it was not registered locally, it will necessarily have already been registered by the partner.)

All subroutines and functions that accept a format string as an argument (with the exception of **DDERegisterFormat**) will test the format string to make sure that it has been registered. The format may have been registered locally, or by the partner program. Most applications will register *all* formats in which they are interested when they start up. Duplicate format registration causes no problems.

Note: EASEL OS/2 EE programs should be careful not to participate in any exchange that requires a non-text based format. Many programs use text formats with names other than DDEFMT_TEXT.

Example:
```
# Register a text-based format
response to start
   copy "Csv" to Format    # Comma separated value forma
   call DDERegisterFormat (Format)
```

1.4.8.2 DDEGetPID Function

Called by: Server

Action: Returns the OS/2 process ID associated with the current instance of EASEL OS/2 EE.

Declaration:
```
function DDEGetPID ()
                    returns integer library "esldde"
```

Remarks: This function is useful when a server wants to register an instance-specific application name so that a client can request that a conversation be established with a particular server instance. Applications typically append their OS/2 process ID to their name (for example, "EASELnnn," where "nnn" is the process ID), and register that name through **DDERegisterServer**. Usually, the WildCard argument to **DDERegisterServer** would be **false** when registering an instance-specific name, since the server would only want to accept conversations for this application name when the client asked for it specifically.

Example:
```
response to start
   copy "Easel" DDEGetPID () to ServerName
   copy "MyTopic" to Topic
   copy false to WildCard
   call DDERegisterServer (ServerName, MyTopic, WildCard)
```

1.4.8.3 DDEGetErrorString Function

Called by: Client or Server

Action: Accepts an error identifier (ID) and returns the error string associated with that ID.

Declaration:
```
function DDEGetErrorString (integer:ErrorID)
                    returns string library "esldde"
```

Where: ErrorID is an error identifier obtained from a call to **DDEGetErrorID**.

Remarks: The error string is a textual description of the error, which is the same one written to the error log when the error occurs. (See 1.5.4, "Recognizing DDE Errors" on page 1-82.)

Example:
```
response to stimulus DDE DDE_ERROR
   copy eventparam to ConvID
   copy DDEGetErrorID to ErrorID
   copy DDEGetErrorString (ErrorID) to ErrorString
```

1.4.8.4 DDESetTraceLevel Subroutine

Called by: Client or Server

Action: Sets the trace level so that you can see a trace of the DDE messages and/or the DDE DLL's actions when you are developing DDE applications.

Declaration:
```
subroutine DDESetTraceLevel (integer:Level)
                    library "esldde"
```

Where: Level is 1, 2, or 3 as follows:

Level #	Description
1	Traces all PM DDE messages that the DDE DLL receives.
2	Traces the DDE DLL's default actions, such as sending WM_DDE_ACK messages.
3	Turns on both Level 1 and Level 2.

Remarks: The trace messages are sent to the EASEL OS/2 EE errorlog. Generally, this is intended to be used as a debugging aid. The trace level should not be set in production EASEL OS/2 EE code.

When the DDE stimulus library is first loaded, it searches the OS/2 environment for a variable called "ESLDDE". If this is found, the trace level will be set to the value of this variable (which should be 1, 2, or 3). The value of the ESLDDE variable is set by using the OS/2 set command. This provides a way to turn on tracing in an EASEL OS/2 EE program that is already compiled. Subsequent calls to **DDESetTraceLevel** will override the initial setting obtained from the environment.

Example:
```
response to start
   copy 3 to TraceLevel
   call DDESetTraceLevel (TraceLevel)
```

1.5 The DDE Events

1.5.1 DDE Default Processing

The EASEL OS/2 EE DDE support includes an EASEL OS/2 EE include file that contains constant definitions for the 12 possible EASEL OS/2 EE DDE events. Whenever the EASEL OS/2 EE DDE DLL receives a DDE message, it notifies the EASEL OS/2 EE program by invoking the corresponding stimulus event (if the program contains one).

- If the EASEL OS/2 EE program contains an active response that matches that event, and the action statements for that response carry out the appropriate DDE protocol response (by calling a DDE DLL function or subroutine), the DDE DLL will not perform any default processing.
- If the EASEL OS/2 EE program does not carry out the correct response, the DDE DLL will perform some appropriate response in order to maintain the integrity of the conversation. Usually, this is to send a negative acknowledgment.

Each EASEL OS/2 EE DDE stimulus event corresponds to an OS/2 PM DDE message, with the exception of the EASEL OS/2 EE DDE DDE_ERROR event, which is invoked when the DDE DLL detects an error.

There are two DDE events that correspond to a single DDE message: DDE_ACK and DDE_NACK. These both correspond to the WM_DDE_ACK message. Normally, a DDE partner sends a WM_DDE_ACK message with a status bit that tells whether the acknowledgment is a positive acknowledgment or a negative acknowledgment. The DDE DLL looks at this status bit and invokes the following:

Acknowledgment	EASEL OS/2 EE Event Invoked	PM Message
Positive	DDE_ACK	WM_DDE_ACK with DDE_FACK status bit set
Negative	DDE_NACK	WM_DDE_ACK with DDE_FACK status bit cleared

This eliminates the need to query the status flag to determine the outcome of every acknowledgment.

1.5.2 Summary of DDE Events

The following table summarizes the DDE events. Following the summary, each event is discussed in detail.

Event	See Page	Description
DDE_INITIATE	1-64	Invoked in a server as notification that the DDE DLL has accepted a conversation on its behalf.
DDE_INITIATEACK	1-65	Invoked in a client as notification that a conversation has been established as a result of a previous call to **DDEInitiate**.
DDE_REQUEST	1-66	Invoked in a server as notification that a client is requesting data.
DDE_ACK	1-68	Invoked in a client or a server when a positive acknowledgment has been received from the partner.
DDE_NACK	1-69	Invoked in a client or a server when a negative acknowledgment has been received from the partner.
DDE_DATA	1-70	Invoked in a client when data has arrived from a server.
DDE_ADVISE	1-72	Invoked in a server when a client wants to establish an advise state.
DDE_UNADVISE	1-75	Invoked in a server when a partner wants to terminate a previously established advise state.
DDE_POKE	1-77	Invoked in a server when a partner has sent unsolicited data.
DDE_EXECUTE	1-79	Invoked in a server when a client requests that one or more commands be executed.

Event	See Page	Description
DDE_TERMINATE	1-80	Invoked in a client or a server when a partner terminates a conversation.
DDE_ERROR	1-81	Invoked in a client or a server whenever the DDE DLL detects an error.

1.5.3 DDE Events List

This section describes each DDE event as follows:

- In which partner the DDE event is invoked.
- The OS/2 PM message received from the partner.
- The meaning of the DDE event.
- The response to that event required by the DDE protocol.
- A description of the DDE default processing.
- The DDE DLL queries available at the time the event is invoked.
- An example.

For each DDE event, **eventparam** is ConvID (the conversation identifier).

1.5.3.1 DDE_INITIATE

Invoked in: Server

PM Message Received from Partner: WM_DDE_INITIATE

Remarks: This event is invoked in a server as notification that the DDE DLL has accepted a conversation on its behalf. (Note that this implies that the EASEL OS/2 EE program has called the **DDERegisterServer** subroutine.) The DDE DLL invokes one of these for each conversation that it accepts on behalf of the server.

Default Processing: The default processing for this event is to accept the conversation if the requested application/topic name matches a pair registered by the server. At the time this event is invoked, the default processing has already been carried out. The EASEL OS/2 EE program need not do any DDE protocol-related processing.

Available Queries: **DDEGetAppName**
DDEGetTopic

Example:

```
response to stimulus DDE DDE_INITIATE
   copy eventparam to ConvID[NumConvs]
   copy (NumConvs + 1) to NumConvs
```

1.5.3.2 DDE_INITIATEACK

Invoked in: Client

PM Message Received from Partner: WM_DDE_INITIATEACK

Remarks: This event is invoked in a client as notification that a conversation has been established as a result of a previous call to **DDEInitiate**. This event is included merely for consistency and can be safely ignored by a client. The fact that a conversation has been established will already be apparent to the application through the **DDEInitiate** return value.

Available Queries: **DDEGetAppName**
DDEGetTopic

Example:

```
response to stimulus DDE DDE_INITIATEACK
   copy eventparam to ConvID[NumConvs]
   copy (NumConvs + 1) to NumConvs
```

1.5.3.3 DDE_REQUEST

Invoked in: Server

PM Message Received from Partner: WM_DDE_REQUEST

Remarks: This event is invoked in a server as notification that a client is requesting data. The server should get the item name (**DDEGetItemName**) and the format (**DDEGetFormat**) to determine if it can honor the request for this item in this format.

Required Response: Upon receipt of a request for data, the DDE protocol requires that the server respond as follows. The server must respond to this event before attempting to transmit another event to the partner.

Acknowledgment	**Subroutine Called**	**PM Message**
Positive (Request can be honored)	**DDEDataString**	WM_DDE_DATA
Negative (Request cannot be honored)	**DDEAck** with DDE_FACK status flag not set	WM_DDE_ACK with DDE_FACK status bit cleared

Default Processing: The DDE DLL default processing is to send a negative acknowledgment if the EASEL OS/2 EE program does not send any acknowledgment.

Available Queries:
DDEGetAppName
DDEGetTopic
DDEGetItemName
DDEGetStatus
DDEGetFormat
DDEGetErrorID

Example:

```
response to stimulus DDE DDE_REQUEST
   copy eventparam to ConvID
   copy DDEGetItemName (ConvID) to ItemName
   copy DDEGetFormat (ConvID) to Format
   if (GetTheData (ItemName, Format, DataString) =
     true) then
      copy (length of DataString) to DataLength
      copy DDESetStatusFlag (0, DDE_FACKREQ) to Status
      call DDEDataString (ConvID, ItemName, DataString,
                          DataLength, Format, Status)
   else
      copy 0 to Status            # send a negative ACK
      call DDEAck (ConvID, ItemName, Format, Status)
   end if
```

1.5.3.4 DDE_ACK

Invoked in: Client or Server

PM Message Received from Partner: WM_DDE_ACK with DDE_FACK status bit set

Remarks: This event is invoked in a client or a server when a positive acknowledgment has been received from the partner. If the application expects multiple acknowledgments, it may want to query the ItemName in order to determine which request is being acknowledged.

Default Processing: The DDE DLL performs no default processing for this event.

Available Queries:
DDEGetAppName
DDEGetTopic
DDEGetItemName
DDEGetStatus
DDEGetFormat
DDEGetErrorID

Example:

```
response to stimulus DDE DDE_ACK
   copy true to HotLinkOK
```

1.5.3.5 DDE_NACK

Invoked in: Client or Server

PM Message Received from Partner: WM_DDE_ACK message with the DDE_FACK status bit cleared

Remarks: This event is invoked in a client or a server when a negative acknowledgment has been received from the partner. Some applications may want to retry the request that resulted in the negative acknowledgment using a different format.

Some partner applications will set the status flags for DDE_FNOTPROCESSED (the last message was not understood) or DDE_FBUSY (the partner rejected the request because it was busy). The application may want to query these status flags.

Available Queries:
DDEGetAppName
DDEGetTopic
DDEGetItemName
DDEGetStatus
DDEGetFormat
DDEGetErrorID

Default Processing: The DDE DLL performs no default processing for this event.

Example:

```
response to stimulus DDE DDE_ACK
   copy true to HotLinkOK
```

1.5.3.6 DDE_DATA

Invoked in: Client

PM Message Received from Partner: WM_DDE_DATA

Remarks: This event is invoked in a client when data has arrived from a server.

Required Response: Upon receipt of data from a server, the DDE protocol requires that the client respond as follows if the DDE_FACKREQ status flag is set. (Query the status by calling **DDEGetStatus** to see if this is true.) The client must respond to this event before attempting to transmit another event to the partner.

Acknowledgment	**Subroutine Called**	**PM Message**
Positive (Data was successfully received)	**DDEAck** with DDE_FACK status flag set	WM_DDE_ACK with DDE_FACK status bit set
Negative (Data was not successfully received)	**DDEAck** with DDE_FACK status flag not set	WM_DDE_ACK with DDE_FACK status bit cleared

Advise State Notes: If the client has one or more active advise states, it should query the status (call **DDEGetStatus**) to see if the DDE_FRESPONSE status flag is set. If it is set, this data is offered in response to a previous advise request.

If this data is the result of an advise state and the DDE_FNODATA status flag was set when the advise state was established, then the client should not attempt to retrieve the data through **DDEGetStringData**. (This will result in an error.) If the client wants the data, it can request it by calling **DDERequest**.

If the DDE_FNODATA status flag was not set, data was sent. It must be retrieved *now* by calling **DDEGetStringData**.

Default Processing: The DDE DLL default processing is to send a positive acknowledgment if the EASEL OS/2 EE program does not send an acknowledgment.

Available Queries:

DDEGetAppName
DDEGetTopic
DDEGetItemName
DDEGetStatus
DDEGetFormat
DDEGetErrorID

Example:

```
response to stimulus DDE DDE_DATA
   copy eventparam to ConvID
   copy DDEGetStringData (ConvID) to StringData
```

1.5.3.7 DDE_ADVISE

Invoked in: Server

PM Message Received from Partner: WM_DDE_ADVISE

Remarks: This event is invoked in a server when a client wants to establish an advise state. An advise state is one where the server notifies the client whenever it recognizes that the data that is the subject of the advise has changed. This is sometimes referred to as a hot link. An application should retrieve the item name (**DDEGetItemName**) and format (**DDEGetFormat**) to determine if the requested item is available in the requested format.

Required Response: Upon receipt of a request to establish an advise state, the DDE protocol requires that the server respond as follows. The server must respond to this event before attempting to transmit another event to the partner.

Acknowledgment	**Subroutine Called**	**PM Message**
Positive (Advise state can be established)	**DDEAck** with DDE_FACK status flag set	WM_DDE_ACK with DDE_FACK status bit set
Negative (Advise state cannot be established)	**DDEAck** with DDE_FACK status flag not set	WM_DDE_ACK with DDE_FACK status bit cleared

Advise State Notes:

If the advise state can be established, the server should query the status (call **DDEGetStatus**) to see if the DDE_FACKREQ and/or DDE_FNODATA status flags are set.

- If the DDE_FACKREQ status flag is set, the server is being requested to send all future data that results from this advise event (via **DDEDataString**) to the client with the DDE_FACKREQ status flag set. (See 1.5.3.6, "DDE_DATA" on page 1-70.)
- If the DDE_FNODATA status flag is set, the server is being requested to send all future data that results from this advise (via **DDEDataString**) to the client with a zero-length data portion. This gives the client, upon receipt of the data, the option of requesting the data if it wants to receive it, while remaining notified of data changes. (See 1.5.3.6, "DDE_DATA" on page 1-70.)

It is the server's responsibility to test for these status flags and honor the requests when sending data to the partner as part of an established advise state.

Default Processing:

The DDE DLL default processing is to send a negative acknowledgment if the EASEL OS/2 EE program does not send an acknowledgment.

Available Queries:

DDEGetAppName
DDEGetTopic
DDEGetItemName
DDEGetStatus
DDEGetFormat
DDEGetErrorID

Example:

```
response to stimulus DDE DDE_ADVISE
   copy eventparam to ConvID
   copy DDEGetFormat (ConvID) to Format
   copy DDEGetItemName (ConvID) to ItemName
   if (DataAvailable (ItemName, Format) = true) then
      copy DDESetStatusFlag (0, DDE_FACK) to Status
      call DDEAck (ConvID, ItemName, Format, Status)
      copy DDEGetStatus (ConvID) to Status
      if (DDEQueryStatusFlag (Status, DDE_FNODATA) =
        true) then
        copy true to NoData
      else
        copy false to NoData
      end if
      if (DDEQueryStatusFlag (Status, DDE_FACKREQ) =
        true) then
        copy true to AckReq
      else
        copy false to AckReq
      end if
   else
      copy 0 to Status              # Send negative ACK
      call DDEAck (ConvID, ItemName, Format, Status)
   end if
```

1.5.3.8 DDE_UNADVISE

Invoked in: Server

PM Message Received from Partner: WM_DDE_UNADVISE

Remarks: This event is invoked in a server when a partner wants to terminate a previously established advise state. An application should retrieve the item name (**DDEGetItemName**) if it is not certain to which item the UNADVISE request corresponds, and respond with a positive or negative acknowledgment, depending on the result of this test. A zero-length ItemName indicates that the partner wants to terminate all advise states associated with that conversation ID.

Required Response: Upon receipt of a request to terminate an advise state, the DDE protocol requires that the server respond as follows. The server must respond to this event before attempting to transmit another event to the partner.

Acknowledgment	Subroutine Called	PM Message
Positive (Advise state has been terminated)	**DDEAck** with DDE_FACK status flag set	WM_DDE_ACK with DDE_FACK status bit set
Negative (Advise state has not been terminated)	**DDEAck** with DDE_FACK status flag not set	WM_DDE_ACK with DDE_FACK status bit cleared

Default Processing: The DDE DLL default processing is to send a negative acknowledgment if the EASEL OS/2 EE program does not send an acknowledgment.

Available Queries:
DDEGetAppName
DDEGetTopic
DDEGetItemName
DDEGetStatus
DDEGetFormat
DDEGetErrorID

Example:

```
response to stimulus DDE DDE_UNADVISE
   copy true to HotLinkTerminated
   copy eventparam to ConvID
   copy DDEGetFormat (ConvID) to Format
   copy DDEGetItemName (ConvID) to ItemName
   copy DDESetStatusFlag (0, DDE_FACK) to Status
   call DDEAck (ConvID, ItemName, Format, Status)
```

1.5.3.9 DDE_POKE

Invoked in: Server

PM Message Received from Partner: WM_DDE_POKE

Remarks: This event is invoked in a server when a partner has sent unsolicited data. The application can retrieve the item name (**DDEGetItemName**) to determine if it is interested in the data. **DDEGetStringData** can be called to retrieve the data.

Required Response: Upon receipt of unsolicited data, the DDE protocol requires that the server respond as follows. The server must respond to this event before attempting to transmit another event to the partner.

Acknowledgment	Subroutine Called	PM Message
Positive (Data is accepted)	**DDEAck** with DDE_FACK status flag set	WM_DDE_ACK with DDE_FACK status bit set
Negative (Data is not accepted)	**DDEAck** with DDE_FACK status flag not set	WM_DDE_ACK with DDE_FACK status bit cleared

Default Processing: The DDE DLL default processing is to send a negative acknowledgment if the EASEL OS/2 EE program does not send an acknowledgment.

Available Queries:
DDEGetAppName
DDEGetTopic
DDEGetItemName
DDEGetStatus
DDEGetFormat
DDEGetErrorID

Example:

```
response to stimulus DDE DDE_POKE
   copy eventparam to ConvID
   copy DDEGetItemName (ConvID) to ItemName
   copy DDEGetFormat (ConvID) to Format
   if (DataIsAccepatable (ItemName, Format) = true) then
      copy DDESetStatusFlag (0, DDE_FACK) to Status
      call DDEAck (ConvID, ItemName, Format, Status)
      copy DDEGetStringData (ConvID) to StringData
   else
      copy 0 to Status
      call DDEAck (ConvID, ItemName, Format, Status)
   end if
```

1.5.3.10 DDE_EXECUTE

Invoked in: Server

PM Message Received from Partner: WM_DDE_EXECUTE

Remarks: This event is invoked in a server when a client requests that one or more commands be executed. The server is expected to retrieve the data using **DDEGetStringData**, and execute the commands contained therein.

Required Response: Upon receipt of a request that one or more commands be executed, the DDE protocol requires that the server respond as follows. The server must respond to this event before attempting to transmit another event to the partner.

Acknowledgment	Subroutine Called	PM Message
Positive (Command can be executed)	**DDEAck** with DDE_FACK status flag set	WM_DDE_ACK with DDE_FACK status bit set
Negative (Command cannot be executed)	**DDEAck** with DDE_FACK status flag not set	WM_DDE_ACK with DDE_FACK status bit cleared

Default Processing: The DDE DLL default processing is to send a negative acknowledgment if the EASEL OS/2 EE program does not send an acknowledgment.

Available Queries:
DDEGetAppName
DDEGetTopic
DDEGetStatus
DDEGetFormat
DDEGetErrorID

Example:

```
response to stimulus DDE DDE_EXECUTE
   copy eventparam to ConvID
   copy DDEGetStringData (ConvID) to Command
   if (ExecuteCommand (ItemName) = true) then
      copy DDESetStatusFlag (0, DDE_FACK) to Status
      call DDEAck (ConvID, ItemName, Format, Status)
      copy DDEGetStringData (ConvID) to StringData
   else
      copy 0 to Status
      call DDEAck (ConvID, ItemName, Format, Status)
   end if
```

1.5.3.11 DDE_TERMINATE

Invoked in: Client or Server

PM Message Received from Partner: WM_DDE_TERMINATE

Remarks: This event is invoked in a client or a server when a WM_DDE_TERMINATE is received. An application should not respond to any other DDE events for this conversation once this event has been invoked. If this event is a response to a previous call to **DDETerminate** (which the application must determine by consulting a variable that it sets when calling **DDETerminate**), then the application should do nothing. Otherwise it should respond with a call to **DDETerminate**.

Default Processing: The DDE DLL default processing is to send the responding WM_DDE_TERMINATE message if the EASEL OS/2 EE program does not.

Available Queries:
DDEGetAppName
DDEGetTopic
DDEGetErrorID

Example:

```
response to stimulus DDE DDE_TERMINATE
   if (LocalTermination = true)  # Partner is responding
      copy 0 to ConvID           # to OUR term
   else                          # Partner wants to
      copy 0 to ConvID           # terminate, take note
   end                           # and let the DDE DLL
                                 # acknowledge it
```

1.5.3.12 DDE_ERROR

Invoked in: Client or Server

Remarks: This event is invoked in a client or a server whenever the DDE DLL detects an error. In most cases, **eventparam** will contain the conversation ID for which the error was discovered. Sometimes, however, **eventparam** will be 0. This would occur, for instance, when an EASEL OS/2 EE program attempted to use an invalid or nonexistent conversation ID, or attempted to call a routine that requires an ID before any conversations have been established.

Ordinarily, an application would call **DDEGetErrorID** and **DDEGetErrorString**, and report the error to the user.

Available Queries:
DDEGetAppName
DDEGetTopic
DDEGetErrorID

Example:

```
response to stimulus DDE DDE_ERROR
   copy eventparam to ConvID
   copy DDEGetErrorID (ConvID) to ErrorID
   copy DDEGetErrorString (ErrorID) to ErrorString
```

1.5.4 Recognizing DDE Errors

Whenever the DDE DLL encounters an error, it invokes a DDE_ERROR event. Note that if the error is discovered while processing an EASEL OS/2 EE response, the DDE_ERROR response is not invoked until the response that caused the error is finished. The DDE DLL also writes the text of the error message to the EASEL OS/2 EE errorlog. All of the functions that return strings return the string "DDE Error" when an error is encountered.

An EASEL OS/2 EE program can always test the value of the built-in function **errorlevel** after calling a DDE subroutine. If an error was encountered, the DDE DLL returns the error ID, which is always non-zero, and can be turned into a string by calling **DDEGetErrorString**. After a successful subroutine call, **errorlevel** will always be zero.

1.5.5 DDE Error Messages

The DDE error IDs and their associated error strings are as follows:

Error ID	Message
1	Can't create conversation (memory error) An error occurred while trying to establish a conversation; there is either not enough memory or the system was unable to create the PM windows required.
2	Bad DDE state (or bad conversation ID) A function or subroutine was called at a time when that function is unavailable, or else an unknown conversation ID was referenced. This error can occur when calling functions or subroutines in the DDE library from a program which has not declared the library as a stimulus DLL, when attempting to access unknown (or improperly initiated) conversations, or when using an invalid conversation ID.
3	Bad subroutine or function argument A subroutine or function has been called with an argument which is out of range. The documentation describes the allowable values for all function and subroutine arguments.

Error ID	Message
4	An inappropriate DDE protocol response was attempted The DDE protocol prescribes what responses are appropriate for each DDE event. This error occurs when a subroutine or function was called which is inappropriate for the event; the calling program is not following the protocol.
5	A partner referenced an unregistered DDE format A partner has erroneously referenced a DDE format which has not been registered with the system.
6	A memory allocation error occurred A general memory allocation error occurred.
7	Error posting DDE message An attempt to post a DDE message to the partner has failed. Either the partner has abnormally terminated, or the system queue is full.
8	An unregistered DDE format was referenced The local program referenced an unknown DDE format. Register the format using DDERegisterFormt.
9	A DDE partner did not respond to a previous termination A DDE conversation partner from a previous conversation has failed to acknowledge termination.
10	An invalid application or topic was referenced This error occurs when **DDEUnRegisterServer** is called with an application or topic name which was not previously registered.
11	An inappropriate attempt to get DDE conversation data was made A function which returns DDE conversation information (status, format, etc.) was called at an inappropriate time. The documentation describes which information is available during each event.
12	An attempt to broadcast a DDE message failed An attempt to initiate a DDE conversation failed. This is a system error.
13	No data was sent by the partner An attempt was made to retrieve DDE data in response to a DDE event which contained no data.

1.6 DDE Support Notes

1.6.1 System Topic

We recommend that all server applications support the system topic. This is a special topic that most servers support to provide a way for a client to interrogate the server on the following:

- What other topics it supports.
- What items can be used with the system topic.
- What formats it understands.

1.6.2 Guarded Blocks

Be careful when using guarded blocks that do not contain DDE stimulus responses, since this can result in the loss of messages and events.

1.6.3 EASEL OS/2 EE DDE Conversation

It is not possible for a particular instance of an EASEL OS/2 EE application to establish a DDE conversation with itself. An EASEL OS/2 EE program can establish a conversation with one or more other instances of itself, however.

1.6.4 Maximum Length of Data

The maximum length of data that can be transferred in a single DDE exchange is (65,522 - (length of ItemName)) bytes.

1.7 DDE Tables

The following tables are included in this section:

- Functions and Subroutines
- DDE Events
- Client Functions/Subroutines and Events
- Server Functions/Subroutines and Events
- PM Messages

1.7.1 Functions and Subroutines

(in alphabetical order)

Subroutine (S) or Function (F)	Called by Client (C) or Server (S)	PM MESSAGE Posted
DDEAck (S)	C, S	WM_DDE_ACK
DDEAdvise (S)	C	WM_DDE_ADVISE
DDEDataString (S)	S	WM_DDE_DATA
DDEEnumerateConversations (F)	C, S	
DDEEnumeratePollTopics (F)	C	
DDEExecute (S)	C	WM_DDE_EXECUTE
DDEGetAppName (F)	C, S	
DDEGetErrorID (F)	C, S	
DDEGetErrorString (F)	C, S	
DDEGetFormat (F)	C, S	
DDEGetItemName (F)	C, S	
DDEGetPID (F)	S	
DDEGetPollAppName (F)	C	
DDEGetPollTopic (F)	C	
DDEGetStatus (F)	C, S	
DDEGetStringData (F)	C, S	
DDEGetTopic (F)	C, S	
DDEInitiate (F)	C	WM_DDE_INITIATE
DDEPokeString (S)	C	WM_DDE_POKE
DDEPollTopics (F)	C	WM_DDE_INITIATE
DDEQueryStatusFlag (F)	C, S	
DDERegisterFormat (S)	C, S	
DDERegisterServer (S)	S	
DDERequest (S)	C	WM_DDE_REQUEST
DDESetStatusFlag (F)	C, S	
DDESetTraceLevel (S)	C, S	
DDETerminate (S)	C, S	WM_DDE_TERMINATE
DDETerminateAll (S)	C, S	WM_DDE_TERMINATE
DDEUnAdvise (S)	C	WM_DDE_UNADVISE
DDEUnRegisterServer (S)	S	

1.7.2 DDE Events

(in alphabetical order)

Event	Invoked in Client (C) or Server (S)	PM Message Received from Partner
DDE_ACK	C, S	WM_DDE_ACK with DDE_FACK status bit set
DDE_ADVISE	S	WM_DDE_ADVISE
DDE_DATA	C	WM_DDE_DATA
DDE_ERROR	C, S	Not Applicable
DDE_EXECUTE	S	WM_DDE_EXECUTE
DDE_INITIATE	S	WM_DDE_INITIATE
DDE_INITIATEACK	C	WM_DDE_INITIATEACK
DDE_NACK	C, S	WM_DDE_ACK with DDE_FACK status bit cleared
DDE_POKE	S	WM_DDE_POKE
DDE_REQUEST	S	WM_DDE_REQUEST
DDE_TERMINATE	C, S	WM_DDE_TERMINATE
DDE_UNADVISE	S	WM_DDE_UNADVISE

1.7.3 Client Functions/Subroutines and Events

(in alphabetical order)

Functions/Subroutines

DDEAck
DDEAdvise
DDEEnumerateConversations
DDEEnumeratePollTopics
DDEExecute
DDEGetAppName
DDEGetErrorID
DDEGetErrorString
DDEGetFormat
DDEGetItemName
DDEGetPollAppName
DDEGetPollTopic
DDEGetStatus
DDEGetStringData
DDEGetTopic
DDEInitiate
DDEPokeString
DDEPollTopics
DDEQueryStatusFlag
DDERegisterFormat
DDERequest
DDESetStatusFlag
DDESetTraceLevel
DDETerminate
DDETerminateAll
DDEUnAdvise

Events

DDE_ACK
DDE_DATA
DDE_ERROR
DDE_INITIATEACK
DDE_NACK
DDE_TERMINATE

1.7.4 Server Functions/Subroutines and Events

(in alphabetical order)

Functions/Subroutines

DDEAck
DDEDataString
DDEEnumerateConversations
DDEGetAppName
DDEGetErrorID
DDEGetErrorString
DDEGetFormat
DDEGetItemName
DDEGetPID
DDEGetStatus
DDEGetStringData
DDEGetTopic
DDEQueryStatusFlag
DDERegisterFormat
DDERegisterServer
DDESetStatusFlag
DDESetTraceLevel
DDETerminate
DDETerminateAll
DDEUnRegisterServer

Events

DDE_ACK
DDE_ADVISE
DDE_ERROR
DDE_EXECUTE
DDE_INITIATE
DDE_NACK
DDE_POKE
DDE_REQUEST
DDE_TERMINATE
DDE_UNADVISE

1.7.5 PM Messages

(in alphabetical order)

PM Message	Function/Subroutine to Call to Transmit This Message to Partner	Event Invoked in EASEL OS/2 EE When Received from Partner
WM_DDE_ACK	**DDEAck**	DDE_ACK (if DDE_FACK status bit is set) DDE_NACK (if DDE_FACK status bit is cleared)
WM_DDE_ADVISE	**DDEAdvise**	DDE_ADVISE
WM_DDE_DATA	**DDEDataString**	DDE_DATA
WM_DDE_EXECUTE	**DDEExecute**	DDE_EXECUTE
WM_DDE_INITIATE	**DDEInitiate** **DDEPollTopics**	DDE_INITIATE
WM_DDE_INITIATEACK	**DDEInitiate**	DDE_INITIATEACK
WM_DDE_POKE	**DDEPokeString**	DDE_POKE
WM_DDE_REQUEST	**DDERequest**	DDE_REQUEST
WM_DDE_TERMINATE	**DDETerminate** **DDETerminateAll**	DDE_TERMINATE
WM_DDE_UNADVISE	**DDEUnAdvise**	DDE_UNADVISE

1.8 DDE Include File

```
#
# include file for esldde.dll
#

function DDEPollTopics (string:AppName, string:Topic)
                                returns integer library "esldde"

function DDEEnumeratePollTopics (integer:LastTopic)
                                returns integer library "esldde"

function DDEGetPollAppName (integer:LastTopic)
                                returns string library "esldde"

function DDEGetPollTopic (integer:LastTopic)
                                returns string library "esldde"

subroutine DDERegisterServer (string:AppName,
                              string:Topic,
                              boolean:WildCard)
                                 library "esldde"

subroutine DDEUnRegisterServer (string:AppName,
                               string:Topic)
                                 library "esldde"

function DDEInitiate (string:AppName,
                      string:Topic)
                                returns integer library "esldde"

function DDEEnumerateConversations (integer: ConvID)
                                returns integer library "esldde"

subroutine DDETerminate (integer:ConvID)
                                library "esldde"

subroutine DDETerminateAll ()
                                library "esldde"

subroutine DDERegisterFormat (string:Format)
                                library "esldde"
```

```
subroutine DDERequest (integer:ConvID,
                       string:ItemName,
                       string:Format,
                       integer:Status)
                                library "esldde"

subroutine DDEDataString (integer:ConvID,
                          string:ItemName,
                          string:DataString,
                          integer:DataLength,
                          string:Format,
                          integer:Status)
                                library "esldde"

subroutine DDEPokeString (integer:ConvID,
                          string:ItemName,
                          string:DataString,
                          integer:DataLength,
                          string:Format,
                          integer:Status)
                                library "esldde"

subroutine DDEExecute (integer:ConvID,
                       string:Command,
                       integer:Status)
                                library "esldde"

subroutine DDEAdvise (integer:ConvID,
                      string:ItemName,
                      string:Format,
                      integer:Status)
                                library "esldde"

subroutine DDEUnAdvise (integer:ConvID,
                        string:ItemName,
                        string:Format,
                        integer:Status)
                                library "esldde"

subroutine DDEAck (integer:ConvID,
                   string:ItemName,
                   string:Format,
                   integer:Status)
                                library "esldde"

function DDEGetStringData (integer: ConvID)
                                returns string library "esldde"
```

```
function DDEGetAppName (integer:ConvID)
                                  returns string library "esldde"

function DDEGetTopic (integer:ConvID)
                                  returns string library "esldde"

function DDEGetItemName (integer: ConvID)
                                  returns string library "esldde"

function DDEGetFormat (integer:ConvID)
                                  returns string library "esldde"

function DDEGetStatus (integer:ConvID)
                                  returns integer library "esldde"

function DDESetStatusFlag (integer:Status,
                           integer:Flag)
                                  returns integer library "esldde"

function DDEQueryStatusFlag (integer:Status,
                            integer:Flag)
                                  returns boolean library "esldde"

function DDEGetErrorID (integer:ConvID)
                                  returns integer library "esldde"

function DDEGetErrorString (integer:ErrorID)
                                  returns string library "esldde"

subroutine DDESetTraceLevel (integer:Level)
                                  library "esldde"

function DDEGetPID ()
                                  returns integer library "esldde"

# values for eventnumber
integer constant DDE_INITIATE     is   1
integer constant DDE_REQUEST      is   2
integer constant DDE_ACK          is   3
integer constant DDE_NACK         is   4
integer constant DDE_DATA         is   5
integer constant DDE_ADVISE       is   6
integer constant DDE_UNADVISE     is   7
integer constant DDE_POKE         is   8
integer constant DDE_EXECUTE      is   9
integer constant DDE_TERMINATE    is   10
integer constant DDE_INITIATEACK  is   11
integer constant DDE_ERROR        is   12


# flags for DDEGet/SetStatus
integer constant DDE_FACK         is   1
integer constant DDE_FBUSY        is   2
integer constant DDE_FNODATA      is   4
integer constant DDE_FACKREQ      is   8
integer constant DDE_FRESPONSE    is   16
integer constant DDE_FNOTPROCESSEDis   32

string constant DDEFMT_TEXT       is   "#1"
```

Index

A

acknowledgments 1-7
advise state 1-2, 1-6
application name 1-5

B

Built-in function 1-3
  eventnumber 1-3
  eventparam 1-3

C

client 1-2, 1-5
client functions/subroutines and events 1-88
client program, sample 1-9
conversation identifier 1-3
conversations
  existing 1-43
  starting 1-23
  terminating 1-40

D

data length 1-84
data, exchanging 1-29
DDE events 1-7—1-8
  client, table 1-88
  server, table 1-89
  table 1-87
DDE functions 1-14—1-17
  client, table 1-88
  server, table 1-89
  table 1-86
DDE include file 1-4, 1-91
DDE partners, client/server 1-5
DDE protocol 1-2
  rules 1-7
DDE stimulus library 1-3
DDE subroutines 1-14—1-17
  client, table 1-88
DDE subroutines *(continued)*
  server, table 1-89
  table 1-86
DDEAck subroutine 1-39
DDEAdvise subroutine 1-34
DDEDataString subroutine 1-31
DDEEnumerateConversations function 1-28
DDEEnumeratePollTopics function 1-20
DDEExecute subroutine 1-33
DDEGetAppName function 1-45
DDEGetErrorID function 1-51
DDEGetErrorString function 1-58
DDEGetFormat function 1-47
DDEGetItemName function 1-48
DDEGetPID function 1-57
DDEGetPollAppName function 1-21
DDEGetPollTopic function 1-22
DDEGetStatus function 1-50
DDEGetStringData function 1-49
DDEGetTopic function 1-46
DDEInitiate function 1-24
DDEPokeString subroutine 1-37
DDEPollTopics function 1-19
DDEQueryStatusFlag function 1-54
DDERegisterFormat subroutine 1-56
DDERegisterServer subroutine 1-25
DDERequest subroutine 1-30
DDESetStatusFlag function 1-53
DDESetTraceLevel subroutine 1-59
DDETerminate subroutine 1-41
DDETerminateAll subroutine 1-42
DDEUnAdvise subroutine 1-36
DDEUnRegisterServer subroutine 1-27
DDE_ACK event 1-61, 1-68
DDE_ADVISE event 1-61, 1-72
DDE_DATA event 1-61, 1-70

DDE_ERROR event 1-62, 1-81
DDE_EXECUTE event 1-79
DDE_FACK event 1-52
DDE_FACKREQ event 1-52
DDE_FBUSY event 1-52
DDE_FNODATA event 1-52
DDE_FNTPROCESSED event 1-52
DDE_FRESPONSE event 1-52
DDE_INITIATE event 1-61, 1-64
DDE_INITIATEACK event 1-61, 1-65
DDE_NACK event 1-61, 1-69
DDE_POKE event 1-61, 1-77
DDE_REQUEST event 1-61, 1-66
DDE_TERMINATE event 1-80
DDE_UNADVISE event 1-61, 1-75
default processing 1-60
dynamic link library (DLL) 1-3, 1-4

E

errors, recognizing 1-82
esldde stimulus library 1-5
ESLDDE.INC include file 1-4, 1-91
eventnumber built-in function 1-3
eventparam built-in function 1-3
events 1-60
  list 1-61
  summary 1-61
exchanging data 1-29
existing conversations 1-43

G

guarded blocks 1-84

H

hot link 1-2, 1-6

I

identifier, conversion 1-3
include file 1-4, 1-91

L

library, stimulus 1-3, 1-5

M

manipulating status variables 1-52
miscellaneous calls 1-55

N

negative acknowledgment 1-7
  notes 1-84

P

Partners, client/server 1-5
PM messages 1-4
  table 1-90
polling for topics 1-18
positive acknowledgment 1-7
programs, sample 1-8
  client 1-9
  server 1-11
protocol rules, DDE 1-7

R

requests for data 1-6
responding to DDE Events 1-7
rules, DDE protocol 1-7

S

sample programs 1-8
server 1-2, 1-5
server functions/subroutines and events 1-89
starting conversations 1-23
status variables, manipulating 1-52
stimulus library 1-3, 1-5

T

terminating a conversation 1-6, 1-40

topic name 1-5
topics, polling 1-18

SC38-7510-00

Printed in U.S.A.